UNIVERSIDAD NACIONAL DE LA AMAZONIA PERUANA

FACULTAD DE INGENIERIA DE SISTEMAS E INFORMATICA

EXPLORACION EN TECNOLOGIAS EMERGENTES:SENSOR GESTUAL "LEAP MOTION"

INFORME PRACTICO DE SUFICIENCIA

PARA OPTAR EL TITULO PROFESIONAL DE:

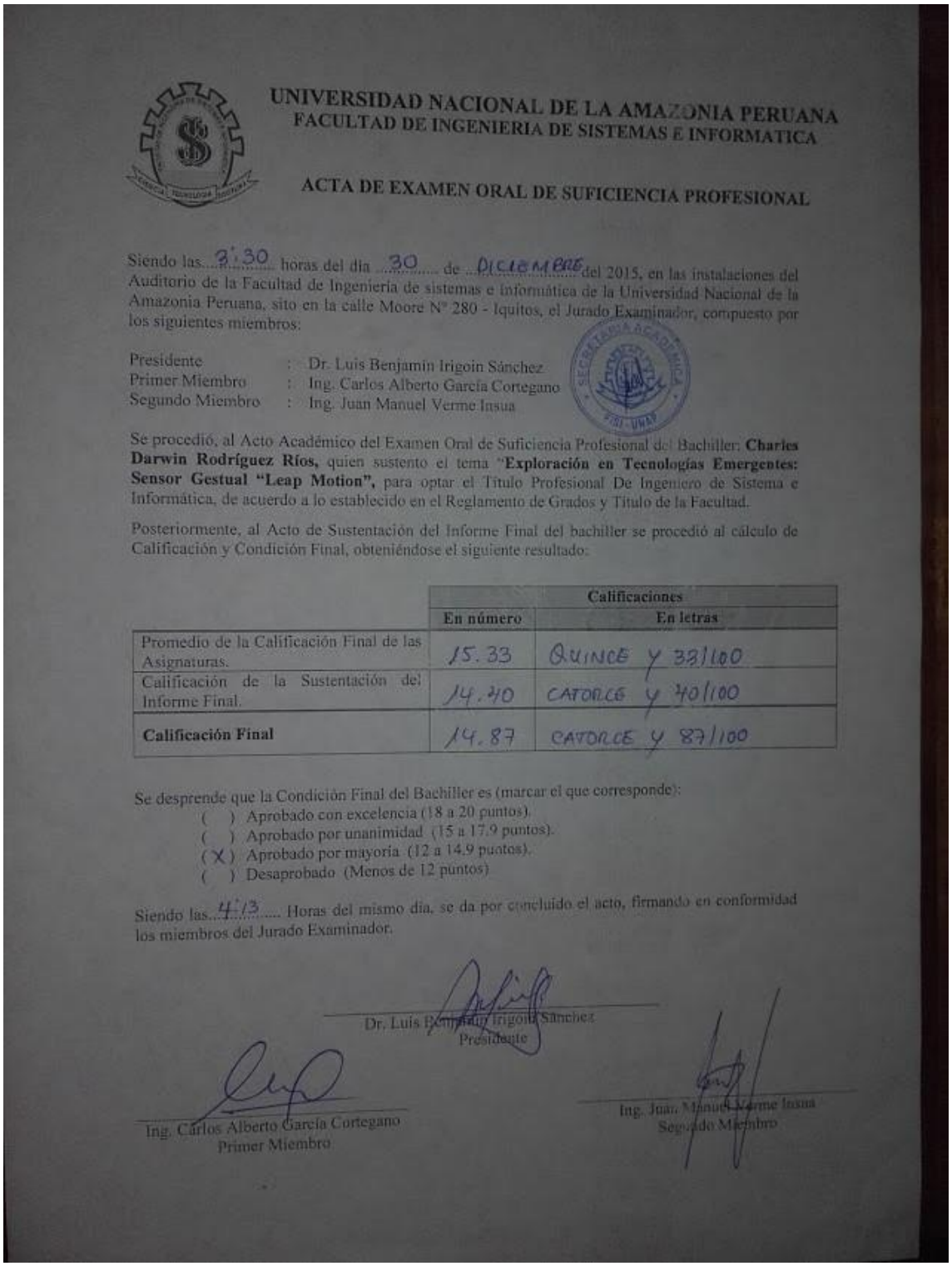
INGENIERO DE SISTEMAS E INFORMATICA

PRESENTADO POR EL BACHILLER:

CHARLES DARWIN RODRIGUEZ RIOS

IQUITOS –PERU

2015



UNIVERSIDAD NACIONAL DE LA AMAZONIA PERUANA
FACULTAD DE INGENIERIA DE SISTEMAS E INFORMATICA

ACTA DE EXAMEN ORAL DE SUFICIENCIA PROFESIONAL

Siendo las 3:30 horas del dia 30 de DICIEMBRE del 2015, en las instalaciones del Auditorio de la Facultad de Ingeniería de sistemas e informática de la Universidad Nacional de la Amazonia Peruana, sito en la calle Moore N° 280 - Iquitos, el Jurado Examinador, compuesto por los siguientes miembros:

Presidente : Dr. Luis Benjamin Irigoin Sánchez
Primer Miembro : Ing. Carlos Alberto García Cortegano
Segundo Miembro : Ing. Juan Manuel Verme Insua

Se procedió, al Acto Académico del Examen Oral de Suficiencia Profesional del Bachiller: **Charles Darwin Rodríguez Ríos,** quien sustento el tema "**Exploración en Tecnologías Emergentes: Sensor Gestual "Leap Motion",** para optar el Titulo Profesional De Ingeniero de Sistema e Informática, de acuerdo a lo establecido en el Reglamento de Grados y Titulo de la Facultad.

Posteriormente, al Acto de Sustentación del Informe Final del bachiller se procedió al cálculo de Calificación y Condición Final, obteniéndose el siguiente resultado:

	Calificaciones	
	En número	**En letras**
Promedio de la Calificación Final de las Asignaturas.	15.33	QUINCE Y 33/100
Calificación de la Sustentación del Informe Final.	14.40	CATORCE Y 40/100
Calificación Final	14.87	CATORCE Y 87/100

Se desprende que la Condición Final del Bachiller es (marcar el que corresponde):

( ) Aprobado con excelencia (18 a 20 puntos).
( ) Aprobado por unanimidad (15 a 17.9 puntos).
(X) Aprobado por mayoría (12 a 14.9 puntos).
( ) Desaprobado (Menos de 12 puntos)

Siendo las 4:13 Horas del mismo dia, se da por concluido el acto, firmando en conformidad los miembros del Jurado Examinador.

Dr. Luis Benjamin Irigoin Sánchez
Presidente

Ing. Carlos Alberto García Cortegano
Primer Miembro

Ing. Juan Manuel Verme Insua
Segundo Miembro

AGRADECIMIENTOS

Durante el desarrollo de este trabajo de fin de carrera son varias las personas a las que les debo mi más sincero y profundo agradecimiento.

A mi familia y mis queridos padres que sin su ejemplo, su esfuerzo y sacrificio no hubiera podido alcanzar este primer objetivo. Siempre han estado a mi lado apoyándome, dándome ánimo y aconsejándome a lo largo de mis estudios y especialmente en los momentos más difíciles de la carrera.

A mi tutor que con su inestimable ayuda, dedicación, disponibilidad, sabiduría, guía y profesionalismo ha permitido que este trabajo tome forma, madure y sea mejor con sus correcciones y sugerencias.

A mis compañeros de carrera y amigos por su colaboración, su amistad y su apoyo incondicional durante este largo recorrido académico.
A todos, ¡muchísimas gracias!

INDICE

INDICE DE FIGURAS

RESUMEN

En los últimos años hubo grandes evoluciones en la forma de interactuar con el computador. Pantallas táctiles sensibles y tecnologías que reconocen el movimiento y la voz dieron lugar a preguntas en cuanto a su eficiencia, naturalidad, intuición y usabilidad.

Por tanto este trabajo tiene como objetivo principal enseñar cómo funciona un elemento (LEAP MOTION) de orden tecnológico que muestra una gran tendencia de innovación y proyectos inimaginables. Busco aquí, inicialmente investigar sobre algunas tecnologías perceptivas capaces de reconocer movimientos, fundamentos en principios de usabilidad de la interacción HUMANO-COMPUTADOR. Con la finalidad de motivar futuros estudios relacionados a este tema.

LEAP MOTION es un Sensor de movimiento con muy alta precisión, Leap Motion Controlador está siendo desarrollado por una pequeña empresa en los Estados Unidos llamada LEAP, fundada en el año 2010. Tal dispositivo fue elegido para ser evaluados en este estudio por su gran la evolución de la cuestión precisamente para conseguir el movimiento del usuario. A través de un nuevo enfoque de reconocimiento, Leap Motion Controlador representado, que no pretende alcanzar un nivel con experiencia en inmersión , la interacción y la navegación con un ordenador.

Las ventajas de este dispositivo según sus creadores, aseguran que la industria química y automotriz sería una de sus grandes destinatarias, la clave del Leap Motion está en la exactitud del diseño o la comprensión de una figura en 3 dimensiones, su configuración es muy sencilla de utilizar.

La página oficial nos ofrece las herramientas necesarias para crear aplicaciones que usen dicho dispositivo, haciendo más fácil su desarrollo y abriendo infinitas posibilidades.

I. INTRODUCCION

1.1.INTRODUCCIÓN AL CONTROL DE MOVIMIENTO

Durante décadas, los controles de movimiento han ocupado un lugar permanente en nuestras visiones del futuro. Hemos visto los súper héroes, científicos locos y vaqueros espaciales de experiencias digitales de control de los medios de comunicación populares con sólo un movimiento de sus manos. Hemos sido cautivados por estas interacciones potentes, naturales, e intuitivas; imaginar lo que sería como tener ese poder en nuestras propias manos. Taller de Tony Stark, "Holodeck" de Star Trek, un escáner cerebral holográfico de Firefly, y las computadoras visionarias pre-crimen de Minority Report todo exuda una sensación de poder y dominio, junto con sentidos paradójicos de la sencillez, la facilidad, la intuición, y la humanidad. Simplemente, estas experiencias se sienten - "Arthur C. Clarke" estilo -magical.

Figura 1 3D holográfica del cerebro escáner en el episodio de Firefly

En esta última década, se han producido do algunos avances asombrosos en tecnologías de consumo que nos acercan, haciendo estas experiencias mágicas en una realidad, pero la tecnología no es la única cosa que debe avanzar para llevar el control de movimiento a la vanguardia. Los patrones de diseño y los paradigmas de software perfeccionadas durante los más de 30 años de ratón, el teclado y el diseño de interacción basado gamepad son sorprendentemente poca utilidad para nosotros, pero la comprensión de su historia es crítica. Tenemos que entender lo que hizo esas interacciones grande, por qué ellos eran amados (y a veces odiado) y cómo podemos crear nuevas e intuitivas, las interacciones de estos dispositivos de entrada nuevos.

a. El desarrollo de interfaces "intuitivos"

Intuitivo es un término peligroso. NUI (Natural User Interface) de hardware, como el controlador de movimiento Leap, a menudo lleva la promesa de interacciones "intuitivos"; pero cuando se pulsa, pocos desarrolladores o diseñadores pueden articular lo que significa ser intuitiva. Cualquiera que haya usado una interfaz de control de movimiento mal diseñado puede decir que el simple uso de los movimientos del cuerpo para controlar el software no produce una experiencia "intuitiva" o "natural". Es mejor romper el término abajo y definir lo que realmente significa ser intuitiva.

Al describir una experiencia intuitiva, a menudo se oye que alguien puede "simplemente sentarse y empezar a usarlo". Estas interfaces se pueden aprender.

Una nota final sobre interfaces intuitivas; lo que es "intuitivo" para una persona puede ser totalmente ajeno a otro. Un adolescente nacido en el año 2000 probablemente no ha utilizado un disco compacto, o incluso oído hablar de un disquete. Estas metáforas no van a ser intuitivo para ellos, donde-según esas mismas metáforas son una segunda naturaleza para un público de más edad. En el diseño de una interacción intuitiva, asegúrese de que primero hay que entender que usted está diseñando para; de lo contrario su capacidad de crear una experiencia intuitiva se reducirá a un golpe de suerte.

Así que ahora tenemos una mejor comprensión de cómo hacer novela, interfaces intuitivas naturales. Para ser intuitivo, nuestras interfaces deben ser:

1. Se puede aprender
2. Comprensible
3. Habitual

Más allá de la intuición, hay una serie de factores que afectarán drásticamente su experiencia de aplicación. Cosas como la ergonomía, diseño de la información, y la detección gesto fiable

b. Ergonomía, Postura, y Medio Ambiente

Una de las primeras cosas en tener en cuenta es el medio ambiente en que va a ser utilizado. Presumiblemente, su solicitud será utilizada por las personas, y en total, somos criaturas bastante limitadas. No nos gusta la celebración de nuestros brazos durante largos períodos de tiempo; y aunque somos muy adeptos a usar nuestras manos, no podemos mantenerlos perfectamente quieto, o hacer traducciones geométricas perfectas. Tenemos la tendencia a moverse y el gesto de diferentes maneras, a veces sutil, a veces desagradable. A menudo la Eficiencia Comercial para la comodidad, o viceversa, dependiendo de nuestra situación; y rara vez nos tomamos el tiempo para optimizar activamente a nuestros espacios físicos. En otras palabras, normalmente tomamos el camino de menor resistencia.

Dependiendo de la intención tendrá que tomar diferentes factores ergonómicos en cuenta. Si su aplicación está destinada para un uso prolongado, considerar el diseño de todas sus interacciones para que alguien pueda realizar con su codo apoyado en una mesa. Si su diseño necesita manos y los brazos de la gente y en movimiento, diseño con períodos de descanso, y las explosiones cortas de interacción. ¿Dónde la gente usa su aplicación? En su ordenador portátil mientras está sentado en la cama, en una silla en su escritorio, de pie en la cocina. Cada entorno tiene sus propios desafíos, limitaciones y oportunidades.

Considere cuánta tensión crean sus interacciones. ¿Es difícil para alguien a quien abrazar su mano o hacer movimientos de granos muy finos en el aire durante un largo período? Movimientos salvajes sin duda tendrá un efecto fatigante. Diseñar para PC tradicional y móvil, estas son preocupaciones secundarias en el mejor, pero con control de movimiento que son críticos para el éxito de su aplicación.

1.2. PROBLEMÁTICA

1.2.1. DEFINICION DEL PROBLEMA

En los últimos años ha habido una gran evolución en la forma de interactuar con dispositivos electrónicos, pero todavía domina la interacción táctil (mecánica) con el ratón y el teclado como los principales periféricos utilizados. Aunque cada vez más presente en los dispositivos modernos, la interacción táctil utiliza también una interfaz física de dos dimensiones.

De este modo, surgen preguntas en cuanto a la viabilidad del uso de una interfaz tridimensional (LEAP MOTION) para manejar este tipo de dispositivos electrónicos, específicamente ordenadores personales, además de la eficiencia ofrecida por las tecnologías de percepción utilizado para ellos.

1.2.2. SOLUCION DEL PROBLEMA

La solución de los problemas planteados se propone mediante el desarrollo un sistema interactivo basado en gestos. Debido a que es una obra predominantemente técnica, buscó centrarse en la tecnología de percepción y desarrollo de software, sin tener en cuenta elementos secundarios relacionados con el cálculo de los gestos y validación de interfaz utilizada.

VENTAJAS DE LEAP MOTION

- **Creación de experiencias únicas**: El Leap Motion te permite crear experiencias de interacción diferentes y únicas con una computadora, imposibles de recrear con los periféricos convencionales: teclado, mouse, gamepad, entre otros.
- **Precisión aceptable:** La precisión del control del dispositivo mejoró considerablemente con las actualizaciones más recientes del SDK. No obstante, aún tiene un amplio margen para seguir mejorando.
- **Fácil instalación y portabilidad**: Se conecta vía USB en cualquier computadora, tiene soporte para los principales sistemas operativos y por su tamaño puede trasladarse fácilmente.

- **Variedad de Aplicación a tu disposición**: Puedes adquirir varias aplicaciones a bajo costo en su tienda **Airspace**.
- **Gran profesionalismo y buena atención del equipo de trabajo**: Excelente y rápida atención de parte del equipo de revisión de la compañía.

DESVENTAJAS DE LEAP MOTION

- **Calentamiento del dispositivo**: Luego de varias horas de uso intensivo el dispositivo tiende a calentarse y pueden empezar percibirse algunos lags en el manejo.
- **Fallos en el reconocimiento**: Si bien la precisión es aceptable, en ocasiones pueden notarse fallos en el reconocimiento de los dedos cuando pones la mano de forma vertical.
- **Produce cansancio muscular:** Puedes experimentar cansancio muscular en los brazos luego de varias horas de uso intensivo.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Este trabajo tiene como objetivo principal enseñar cómo funciona un elemento (LEAP MOTION) de orden tecnológico que muestra una gran tendencia de innovación y proyectos inimaginables.

1.3.2. OBJETIVOS ESPECIFICOS

- Desarrollar una aplicación que se capaz de funcionar con el dispositivo leap motion.
- Investigar a fondo la documentación sobre la API de leap motion.
- Entender el funcionamiento del código para desarrollar futuras aplicaciones.

METODOLOGIA

El trabajo comenzó con una búsqueda bibliográfica para profundizar conocimientos en el campo de la Interacción Persona-Ordenador e Informática Perceptual.
En este estudio hemos definido el tema, el contexto, las problemáticas, los objetivos generales y específicos y la estructura de la obra.
Estas tecnologías se evaluaron y compararon en su núcleo características, comprobando su funcionalidad para reconocer movimientos.
El desarrollo de la aplicación se inició después de la elección de la tecnología considera más apropiado y su configuración del entorno el desarrollo.

II. FUNDAMENTOS TEORICOS Y TECNOLOGICOS

2.1 INTERACCION HUMANO COMPUTADOR

Según la Sociedad Brasileña de Computación (SBC, 2013), el área Human-Computer Interaction (HCI) se dedica al estudio de los fenómenos de la comunicación entre las personas y los sistemas informáticos que se encuentran en la intersección de ciencias de la computación y la información y las ciencias sociales y del comportamiento, que cubre todos los aspectos relacionados con la interacción del usuario con el sistemas.
La interacción usuario-sistema en marco computacional es la combinación de software y hardware necesarios para facilitar el proceso de comunicación entre el usuario y la aplicación.
Según Norman (Norman, 1986), el término se utiliza comúnmente interfaz para denotar que interconecta dos sistemas. Las viejas interfaces entre los usuarios y los sistemas informáticos son criticadas por esfuerzo del usuario exigente mayor en las actividades cognitivas de interpretación y expresión de la información que los procesos del sistema.
Recientemente ha habido un interés creciente en la introducción de otros medios de interfaz hombre-máquina en el campo. Estos nuevos medios de comunicación incluyen una clase de dispositivos basada en el movimiento o, más específicamente, los gestos.

2.1.1INTERACCION NATURAL

El cambio tecnológico trae la capacidad de interconectar cuya aplicación utilice progresivamente del acto natural del hombre, lo que resulta en una interacción más intuitiva.
Según Garbin (GARBIN, 2010), la expresión interacción natural es estar utilizado por varios estudios en general, o como un perceptual independiente en que describen como algo que es de alguna manera diferente de las interfaces común, visto antes. Se muestra una comparación entre las interfaces en la Figura 2. El uso de este concepto en el desarrollo de interfaces asistencias en buena aceptación de los usuarios, ya que su ligera influencia de asociar lo que son, se trata de algo abstracto, medios digitales, pero con objetos físicos. La Interacción Natural - EN cognición activa y gente dinámica cibernéticos, que por lo general Sucede cuando experimentan algo en la vida real (Medeiros, 2012).

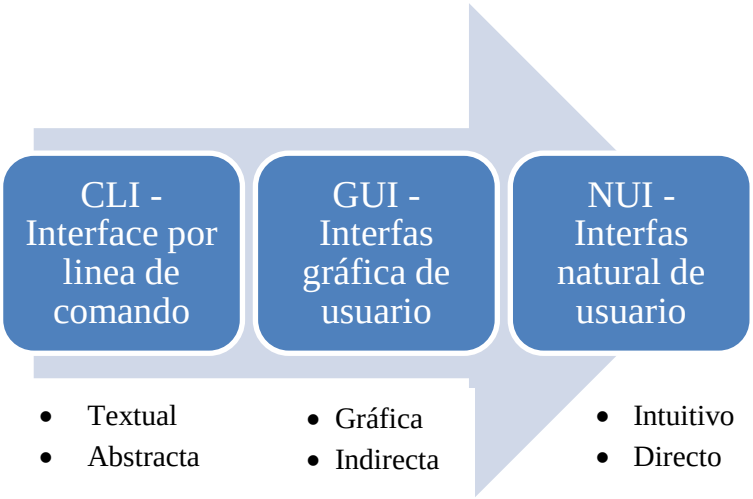


Figura 2. Evolución de las interfaces. Fuente : (GABIN , 2010)

Considerado como un nuevo paradigma de interacción, Interfaz Natural El usuario es un lenguaje común utilizado por los diseñadores y desarrolladores interfaz de la computadora para referirse a una interfaz de usuario que se aplica conceptos de interacción naturales en su construcción.
En teoría, la interacción con una capacidad NUI debe depender solamente usuario interactuar con el medio ambiente. A pesar de que requiere un aprendizaje, la interfaz es diseñado para pasar la sensación de que usted y la máquina está conectado

2.1.2 INTERACCION POR GESTOS

A pesar de sus diferentes significados en este trabajo será la palabra gesto adoptada con la definición propuesta por Mitra y Acharya: " Los gestos son movimientos corporales expresivos y significativos realizado moviendo las manos, brazos, cabeza o el cuerpo con el fin de: 1 transmitir un información o 2 interactuar con el medio ambiente". (S. Mitra y T. Acharya, 2007).
Los gestos son complejos mecanismos para la asignación ya que pueden tener varios significados, que varían de acuerdo a los aspectos contextuales y culturales, También puede variar de acuerdo a la interacción con el medio ambiente. Un gesto, como agitar o aplaudir, no implica ningún elemento extraño, a diferencia de mover, empujar, agarrar, la participación de otros objetos.
Según Ghirotti (Ghirotti, 2009), las nuevas tecnologías son perceptivos un paso para hacer interfaces naturales en una realidad en el uso diario la informática; y, sobre la base de estos conceptos estudiados, se presentan de tres tecnologías aquí elegidos para ser examinados y evaluados.

2.2 TECNOLOGIAS PERCEPTIVAS

Tecnologías de percepción que tienen que ver con la interacción natural, tienen como propósito principal de permitir a los ordenadores a una detección sensorial detallada los seres humanos. Estas tecnologías tienen una gama de hardware y software que Trabajan principalmente con los siguientes puntos de vista: el reconocimiento de gestos, reconocimiento facial, reconocimiento de voz y el reconocimiento táctil.
Uno de los propósitos de este trabajo es que las tecnologías perceptiva, responsable del reconocimiento de gestos tridimensionales. Tres herramientas presentadas que se ocupan de este tipo de reconocimiento por llevar a cabo el estudio y la implementación de la aplicación.
Los dispositivos fueron elegidos Microsoft Kinect ™ para Windows, por extensa documentación disponible y sus múltiples posibilidades de aplicación; Asus Xtion, las características y el potencial similar a Kinect; y el Leap Motion Controlador, innovación propuesto por el concepto y la oportunidad de trabajar con un nuevo enfoque a la tecnología perceptual, en la cual nos centraremos y desarrollaremos nuestra aplicación.

2.2.1 MICROSOFT KINECT PARA WINDOWS

Kinect ™ es un sensor de movimiento desarrollado originalmente por © PrimeSense y adquirida por la empresa Microsoft para ser utilizado como una periférica consola de juegos dispositivo de la misma empresa, llamado Xbox 360 ™. Su objetivo principal es permitir a los usuarios controlar e interactuar con la

consola a través de una interfaz de usuario natural usando gestos y comandos de voz, sin usar el control manual.

Con el éxito de las ventas y el aumento de su popularidad, Microsoft decidido invertir en otras aplicaciones para ello con el fin de explorar la mayor parte de su potencial (WIKI KINECT, 2013).Luego, en febrero de 2012, Microsoft lanzó Kinect para Windows consiste en kit de desarrollo de sensor Kinect ™ y para uso comercial necesario para desarrollar aplicaciones en el sistema operativo Windows. El SDK de Kinect ™ es compatible con las aplicaciones escritas en C ++, C # o Visual Basic5 utilizando la llamada IDE de Microsoft Visual Studio (MICROSOFT, 2013).

El sensor Kinect ™ tiene 3 cámaras: a RGB convencional y dos infrarrojos utiliza para detectar la distancia del usuario, y un conjunto de micrófonos utilizados para recoger el habla, como se muestra en la Figura 3.

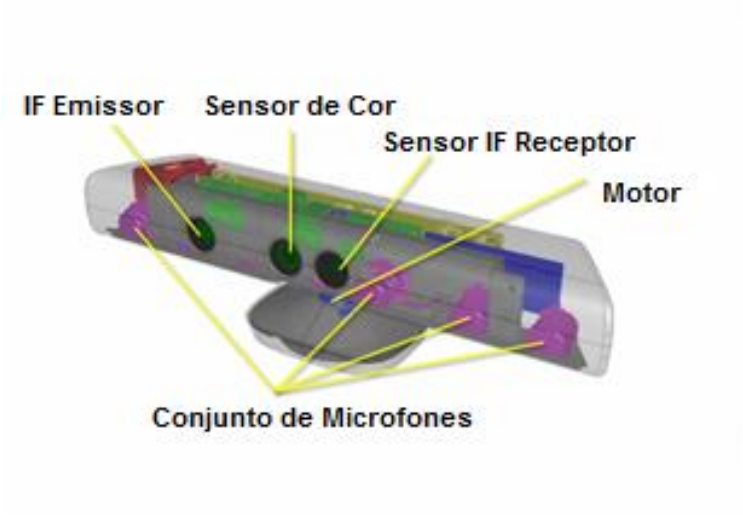


Figura 3. Estructura de Kinect. Fuente: (KINECT, 2013)

Sus características y capacidades son (MICROSOFT, 2013):

- Cerca de 23 cm de longitud horizontal, longitud de 5 cm verticales y 5 cm.
- Ángulo de 43,5º y un motor de visualización vertical para cambiar el ángulo visión + -27º y 27º en vertical. La Figura 4 se representa el ángulo de visión de la Kinect.

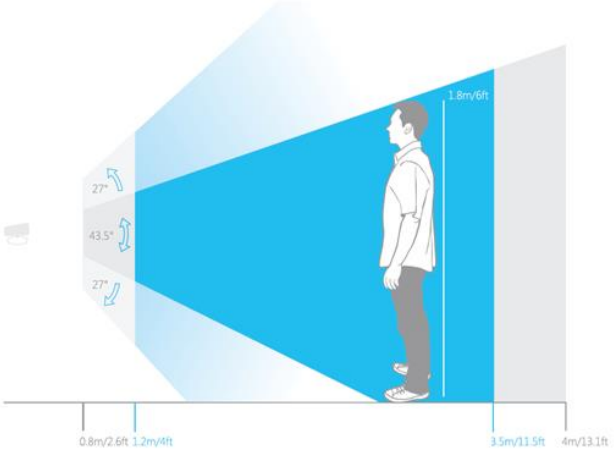


Figura 4. Campo de visión de Kinect. Fuente: (KINECT, 2013)

- Cuenta con una cámara RGB (rojo, verde, azul) que se lleva a 30 cuadros por segundo en una resolución de 640x480 y 15 cuadros por segundo en resolución 1280x1024.
- Un emisor de infrarrojos y una profundidad del sensor IR. El emisor emite haces de luz infrarroja y el sensor de profundidad lee las vigas infrarroja reflejada de nuevo al sensor. Los haces reflejados son convertido a la información de profundidad, generando distancia entre objeto y el sensor.
- Tiene un conjunto de micrófonos incorporados, que además de la captura de las voces más cercanas, se pueden diferenciar el ruido externo.
- Un acelerómetro de 3 ejes configurado para una gama de 2g donde g es la aceleración debida a la gravedad.
- Ha incorporado en el procesador.
- Utilice una interfaz USB 2.0 para la comunicación con el ordenador.
- Detecta 20 puntos de articulación del esqueleto humano, conocido como articulaciones.

A diferencia de Kinect ™ para la consola de juegos, el Kinect para Windows cuenta con un sensor de ángulo de visión de cerca, lo que le permite ponerse al día diez articulaciones en la parte superior del cuerpo humano, y también cuando el usuario se percate está sentado, una gran ventaja para trabajar al solicitar un pantalla del usuario cerca de la computadora.
En la Figura 5, el sensor Island diferencia de que sólo tiene el modo por defecto, y el sensor para Windows, que además del método anterior, tiene el modo Near.

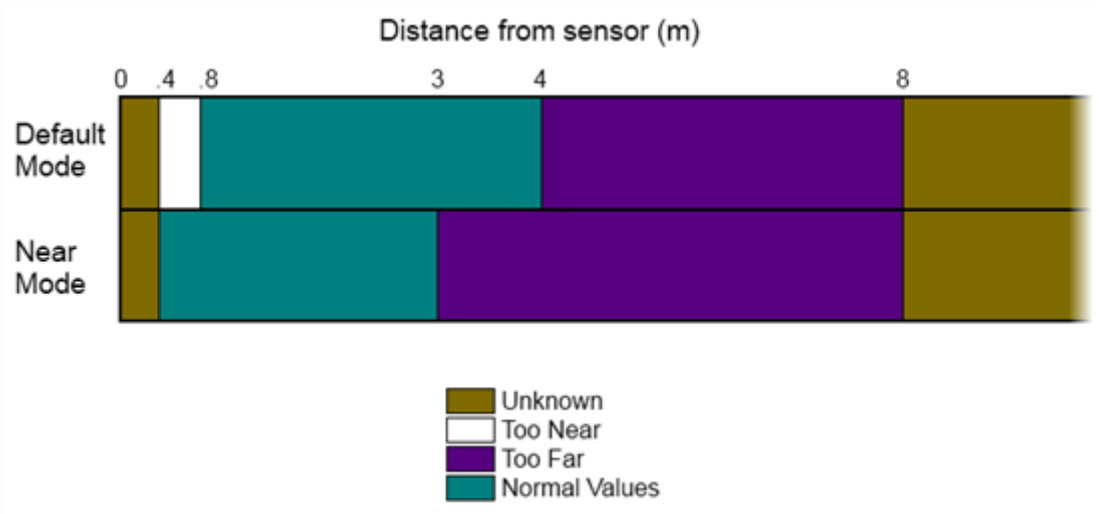


Figura 5 Comparación entre modos de distancia de la Kinect. Fuente: (KINECT MODOS, 2013)

El Kinect tiene un esqueleto sistema de seguimiento (SR) que proporciona las posiciones de las articulaciones. Estas posiciones se proporcionan como coordenadas en un eje tridimensional y usado para muchos propósitos, tales como detección de gestos, la navegación, la manipulación de objetos tridimensionales, entre otros.

Según Azimi (Azimi, 2013), en el uso práctico del sensor, un ruido aparece en las coordenadas de las articulaciones devueltos por este sistema. Muchos parámetros afectan a las características y el nivel de este ruido, como la iluminación de la habitación, tamaño del cuerpo del usuario, la distancia entre el usuario y el sensor, la posición del usuario, ubicación del sensor y de redondeo efectos introducidos por cálculo.

Azimi dijo que las posiciones de las articulaciones devueltos por el sistema tienen exactitud, es decir, poca diferencia en las coordenadas recibidas a la posiciones efectivas en el mundo real. Sin embargo, las posiciones de los datos de las articulaciones no son necesariamente exactas, variando dentro de la casa dos centímetros.

También menciona que hay casos en que el sistema de seguimiento no tiene información suficiente para determinar la posición específica de una articulación, tal como oclusión moviendo o por otros, auto- oclusión de un conjunto de articulaciones y mover una parte del cuerpo fuera del campo de visión del sensor, entre otros. Sino en la mayoría de los casos, el MR es capaz de inferir la posición de la articulación. Por lo tanto, dos tipos de ruido están presentes en la traza de las articulaciones: un ellos es relativamente pequeño y está siempre presente en todas las mediciones, causado por la imprecisión del sensor; la otra son causados por cambios temporales por inexactitud cuando una articulación tiene un estado de seguimiento indeterminado.

2.2.1.1 MICROSOFT KINECT SDK

La implementación del sistema y la comunicación con el sensor Kinect ™ puede se realizará con diferentes kits de desarrollo (SDK). Microsoft tiene su propio kit estable es la versión 1.7, que tiene una amplia documentación e integra todo el proceso de instalación y configuración del sensor de los conductores.

El SDK proporciona acceso a las interfaces de programación de aplicaciones (API) Kinect ™, lo que permite las siguientes características (Microsoft, 2013):

- control directo del sensor;
- Acceso directo a las imágenes de sensores de profundidad , y RGB;
- Acceso al micrófono con capacidades de procesamiento de audio, incluyendo la supresión de ruido y cancelación de eco;
- Acceso a los datos del acelerómetro;
- Detección del esqueleto del usuario en el campo de Kinect de vista gestos aplicaciones orientadas;
- Integración con la API de reconocimiento de voz de Windows;
- Apoyar a la última versión del sistema operativo de Windows (8.1);
- Apoyar la última versión de Microsoft IDE (2012); y
- Creación de aplicaciones en C ++, C #, Visual Basic.

2.2.2 ASUS XTION

En primer lugar exclusivo sensor de movimiento comercial a la PC (ASUS, 2013) el Xtion es un dispositivo electrónico distribuido por ASUS, la compañía de tecnología Taiwán que utiliza sensores de tipo infrarrojos para detectar la profundidad, RGB para detectar colores, y un conjunto de micrófonos para capturar el seguimiento de voz en tiempo real de los movimientos realizados por el usuario.
El dispositivo se muestra en la Figura 6, viene con un conjunto de herramientas desarrollo para que sea más fácil para los desarrolladores crear sus propias aplicaciones basado en gestos, sin la necesidad de escribir algoritmos de programación complejos.

Figura 6 Xtion estructura. Fuente: (ASUS, 2013)

Según Asus, es capaz de seguir todo el cuerpo y sus movimientos, por lo que es ideal para su uso en juegos como control. El Xtion soporta reconocimiento a diferentes sistemas operativos multiusuario.

2.2.3 LEAP MOTION CONTROLLER

Sensor de movimiento con muy alta precisión, Leap Motion Controlador está siendo desarrollado por una pequeña empresa en los Estados Unidos MOVIMIENTO llamada LEAP, fundada en el año 2010. Tal dispositivo fue elegido para ser evaluados en este estudio por su gran la evolución de la cuestión precisamente para conseguir el movimiento del usuario. A través de un nuevo enfoque de reconocimiento, Leap Motion Controlador representado en la Figura 7 , que no pretende alcanzar un nivel con experiencia en inmersión , la interacción y la navegación con un ordenador.

Figura 7. Funcionamiento y arquitectura de Leap Motion fuente:(Leap Motion, 2015)

2.2.3.1 ¿QUÉ ES LEAP MOTION?

Es un diminuto dispositivo donde se aplican las tecnologías actuales de sensores, que colocado frente a la pantalla de nuestro ordenador es capaz de capturar con una precisión enorme los movimientos de nuestras manos, dedos e incluso objetos.

- Dispositivo de rastreo
- Interacción sin contacto con una computadora
- Alta precisión
- Dispositivo de consumo

2.2.3.2 FUNCIONAMIENTO

El controlador Leap Motion abarca tanto los componentes de hardware y software. El hardware Leap Motion consiste principalmente en un par de infrarrojos estéreo cámaras y LEDs de iluminación. Los sensores de la cámara miren hacia arriba (cuando el dispositivo está en su orientación estándar). La siguiente ilustración muestra cómo las manos del usuario se ven desde la perspectiva del sensor Leap Motion.

El software Leap Motion recibe los datos del sensor y analiza estos datos específicamente para las manos, los dedos, los brazos y las herramientas. (Seguimiento de otros tipos de objetos que se podrán añadir más en el futuro, pero esto es el conjunto actual de las entidades supervisadas). El software mantiene un modelo interno de la mano humana y compara ese modelo a los datos de los sensores para determinar el mejor ajuste. Los datos del sensor se analizan fotograma a fotograma y el servicio envía cada cuadro de datos a las aplicaciones habilitadas para movimiento de salto. El Marco de objeto recibido por la aplicación contiene todas las posiciones conocidas, las velocidades y las identidades de las entidades de orugas, como las manos, los dedos y las herramientas. Las construcciones tales como los gestos y movimientos que abarcan varios marcos también se actualizan cada fotograma. Para una visión general de los datos de seguimiento proporcionados por el controlador.

El software Leap Motion se ejecuta como un servicio (Windows) o demonio (Mac y Linux) en el equipo cliente. Las aplicaciones habilitadas para Motion Nativo Salto pueden conectarse a este servicio mediante la API disponga en las librerías dinámicas Leap Motion (proporcionados como parte del SDK de Leap Motion). Aplicaciones Web pueden conectarse a un servidor WebSocket organizada por el servicio. El WebSocket proporciona datos de seguimiento como un mensaje con formato JSON - un mensaje por trama de datos. Una biblioteca JavaScript LeapJS, proporciona una API envolver estos datos.

2.2.3.3 USOS

- Jugar
- Películas, (los efectos)
- Médico
  - Cirugía
  - Biomecánica
  - Los procedimientos remotos
- Industrial
- Tecnología útil
- Más

2.2.3.4 ESPECIFICACIONES TECNICAS

- Altura :0.5 pulgadas
- Ancho :1.2 pulgadas
- Profundidad :3 pulgadas
- Peso :0.1 libras

Requisitos mínimos del sistema

- Windows 7, 8, Mac OS 10.7 Lion
- Procesador AMD Phenom™ II ó Intel® Core™ i3,i5,i7
- RAM 2 GB
- Puerto USB 2.0
- Conexión a Internet
- Garantía 1 año

2.2.3.5 LEAP MOTION PARA DESARROLLADORES

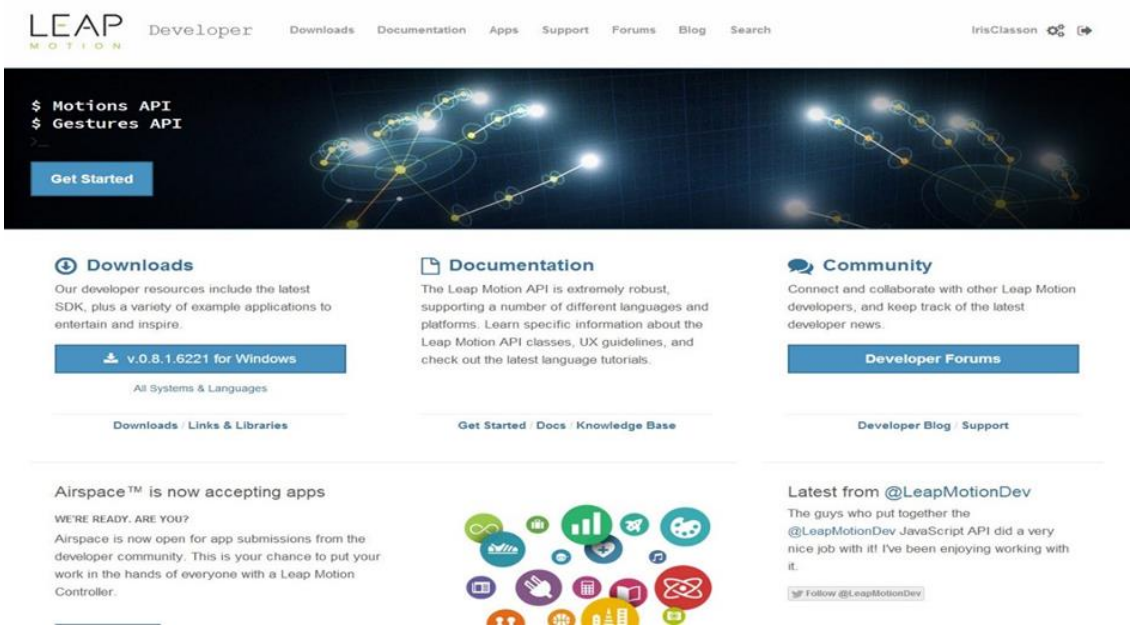


Figura 8. Portal Leap Motion para desarrolladores

2.2.3.6 LENGUAJES SOPORTADOS

- C#
- Java
- C++
- Python
- Javascript

2.2.3.7 API DESCRIPCION

El sistema Leap Motion reconoce y rastrea las manos, los dedos y las herramientas de dedos. El dispositivo funciona en una proximidad íntima con posiciones de alta precisión y velocidad de fotogramas de seguimiento e informes discreta, los gestos y el movimiento.

El controlador Leap Motion utiliza sensores ópticos y luz infrarroja. Los sensores están dirigidos a lo largo del eje y- hacia arriba cuando el controlador está en su posición de funcionamiento estándar -y tienen un campo de visión de aproximadamente 150 grados. El alcance efectivo del Motion Controller Salto se extiende desde aproximadamente 25 a 600 milímetros por encima del dispositivo (1 pulgada a 2 pies).

figura 9 leap motion controlador de tus manos

Detección y seguimiento de obra más cuando el controlador tiene una visión de alto contraste claro de la silueta de un objeto. El software Leap Motion combina sus datos de sensores con un modelo interno de la mano humana para ayudar a hacer frente a las condiciones de seguimiento desafiantes.

2.2.3.7.1 SISTEMA DE COORDENADAS

El sistema Leap Motion emplea un sistema de coordenadas cartesianas diestro. El origen se centra en la parte superior del controlador de movimiento de salto. Los ejes X y Z ejes se encuentran en el plano

horizontal, con el eje x funcionamiento paralelo al borde largo del dispositivo. El eje y es vertical, con el aumento de los valores positivos hacia arriba (en contraste con la orientación hacia abajo de la mayoría de los gráficos por ordenador sistemas de coordenadas). El eje z tiene valores positivos crecientes hacia el usuario.

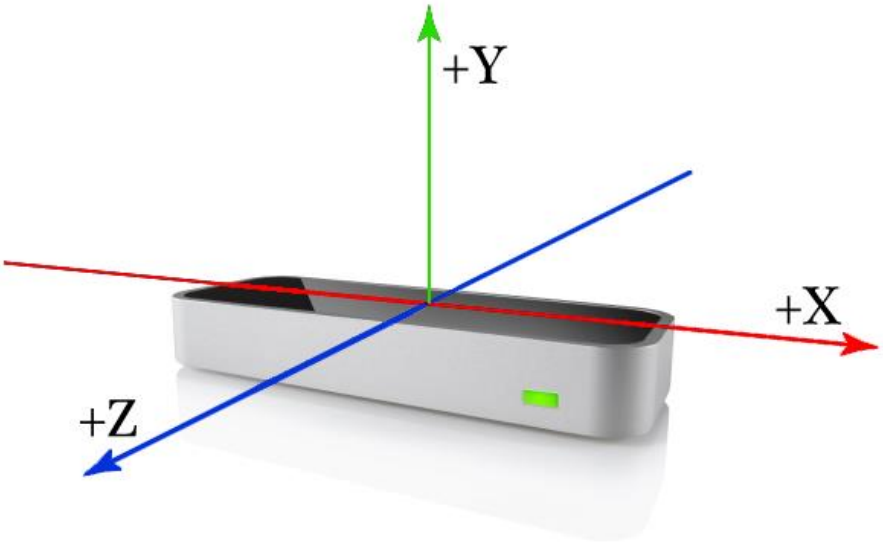


FIGURA 10 El sistema diestro coordinar Leap Motion.

La API de Leap Motion mide cantidades físicas con las siguientes unidades:

Distancia:	milímetros
Hora:	microsegundos (a menos que se indique lo contrario)
Velocidad:	milímetros / segundo
Ángulo:	radianes

2.2.3.7.2 MANOS

El modelo de la mano proporciona información acerca de la identidad, posición y otras características de una mano detectado, el brazo al que se adjunta la mano, y las listas de los dedos asocia con la mano.

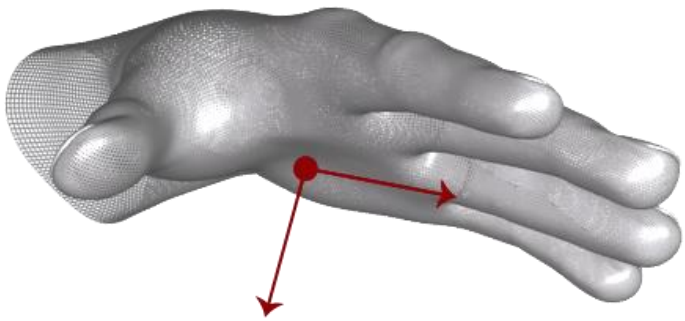


Figura 11. Los Mano Palma Normal y Dirección vectores que definen la orientación de la mano.

2.2.3.7.3 BRAZOS

Un brazo es un objeto similar al hueso que proporciona los puntos de orientación, longitud, anchura, y final de un brazo. Cuando el codo no está a la vista, el controlador Leap Motion calcula su posición sobre la base de las observaciones anteriores, así como la proporción humana típica

2.2.3.7.4 DEDOS

El controlador Leap Motion proporciona información sobre cada dedo en una mano. Si todo o parte de un dedo no es visible, las características de los dedos se estiman basándose en observaciones recientes y el modelo anatómico de la mano. Los dedos se identifican por el nombre del tipo, es decir, el pulgar, índice, medio, anular y meñique.

Los dedos están representados por la Finger class, que es una especie de orientable objeto.

Figura 12 Finger TipPosition y de dirección vectores proporcionan la posición de la punta del dedo y la dirección general en la que un dedo está apuntando.

2.2.3.7.5 HERRAMIENTAS

Una herramienta es un objeto como un lápiz.

Herramientas están representados por la herramienta de clase, que es una especie de orientable objeto.

Figura 13 herramientas

Objetos cilíndricos Sólo delgadas son rastreados como herramientas.

2.2.3.7.6 GESTOS

El software Leap Motion reconoce ciertos patrones de movimiento como los gestos que podrían indicar una intención de usuario o de comandos. Los gestos son observados para cada dedo o una herramienta de forma individual. Los informes de software Leap Motion gestos observados en un marco de la de la misma manera que los informes de otros datos de seguimiento de movimiento como los dedos y las manos.

Los gestos son representados por el gesto de clase y sus subclases, **CircleGesture, KeyTapGesture, ScreenTapGesture y SwipeGesture** Los siguientes patrones de movimiento son reconocidos por el software de Leap Motion:

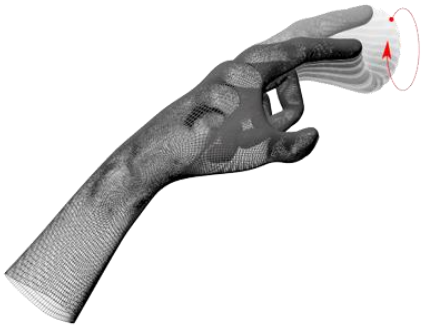


Figura 14. Círculo - un dedo trazando un círculo.

Figura 15. Flagelo - Un largo movimiento lineal de

Figura 16. Toque la tecla - Un movimiento tocando con un dedo como si pulsando una tecla del teclado.

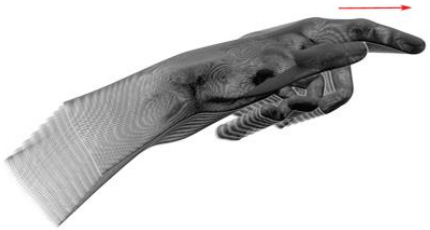


Figura 17. Pantalla Tap - Un movimiento tocando con el dedo como si tocando una pantalla de ordenador vertical.

2.2.3.7.7 MOVIMIENTOS

Los movimientos son estimaciones de los tipos básicos de movimientos inherentes en el cambio de las manos del usuario durante un período de tiempo. Los movimientos incluyen escala, rotación y traslación (cambio de posición).

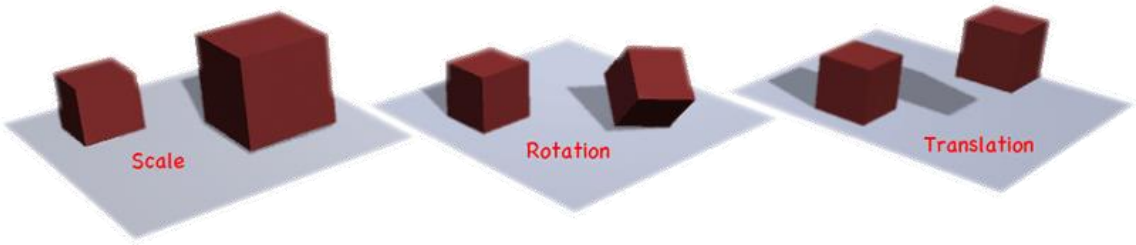


Figura 18. Movimientos

Los movimientos se calculan entre dos cuadros. Usted puede obtener los factores de movimiento para la escena en su conjunto de un marco de objeto. También puede obtener los factores asociados con una sola mano de una mano objeto.

Puede utilizar los factores de movimiento reportados para diseñar las interacciones dentro de la aplicación. Por ejemplo, en lugar de seguir el cambio de posición de los dedos individuales a través de varias tramas de datos, se puede utilizar el factor de escala calculado entre dos marcos de dejar al usuario cambiar el tamaño de un objeto.

Tipo de movimiento	Marco	Mano
Escala	Escalamiento Frame refleja el movimiento de la escena de objetos hacia o lejos el uno del otro. Por ejemplo, con una mano se mueve más cerca de la otra.	Escalamiento Mano refleja el cambio en la propagación dedo.
Rotación	La rotación del marco refleja el movimiento diferencial de los objetos dentro de la escena. Por ejemplo, con una mano arriba y la otra hacia abajo.	Rotación de la mano refleja el cambio en la orientación de una sola mano.
Traducción	Traducción marco refleja el cambio promedio en la posición de todos los objetos de la escena. Por ejemplo, las dos manos se mueven hacia la izquierda, hacia arriba o hacia adelante.	Traducción Mano refleja el cambio en la posición de esa mano.

2.2.3.7.8 SENSOR DE IMÁGENES

Junto con los datos de seguimiento calculados, usted puede obtener las imágenes de sensores primas procedentes de las cámaras Leap Motion.

Los datos de imagen contiene los valores de medición de brillo IR y los datos de calibración requeridos para corregir la distorsión de la lente compleja. Usted puede utilizar las imágenes de sensores para

aplicaciones de realidad aumentada, especialmente cuando el hardware Leap Motion se monta a un auricular VR.

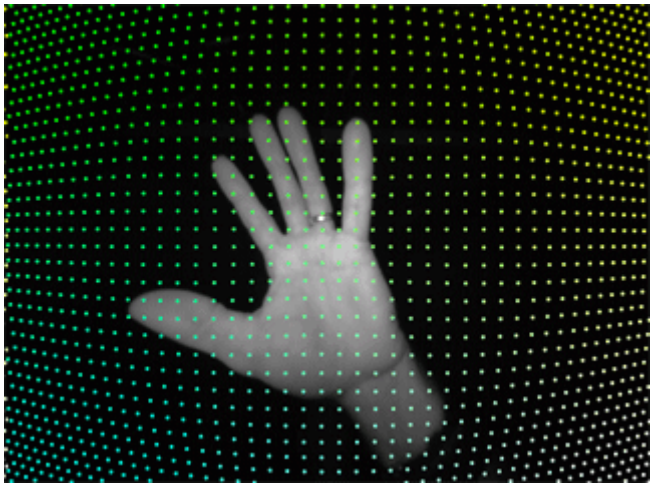


FIGURA19. Imagen del sensor de crudo con puntos de calibración superpuestas.

2.3 COMANDOS DEL COMPUTADOR

Los Ordenadores Actualmente se manejan, principalmente a través de dispositivos de control artificiales, como un ratón o pantallas sensibles Touchpad6 el tacto y el teclado. El propósito de este estudio es explorar un nuevo enfoque, comandos informáticos que permitan se interpretan los gestos.

Así, entre las posibilidades de realizar movimientos con el cuerpo, se consideran algunos de los comandos del sistema operativo Microsoft Windows, basado en la versión 8 (ocho), que se utiliza principalmente la manipulación archivos y ventanas, abrir aplicaciones y la navegación entre páginas y características escritorio.

Entre los comandos que se pueden utilizar son: clic, seleccionar, arrastrar, girar, desplazamiento, el menú desplegable de la pantalla para mostrar el escritorio, navegue entre las ventanas, acercar y zoom, y cerrar, maximizar y minimizar la ventana.

III. DESARROLLO DE LA APLICACIÓN

3.1 MI EXPERIENCIA DESARROLLANDO UN VIDEOJUEGO PARA LEAP MOTION

Desarrollar un videojuego para el Leap Motion es una tarea bastante laboriosa y en ocasiones compleja. Lo más difícil es pensar en cómo adaptar el game play a la estructura de control y detección de gestos que ofrece el SDK. En un principio, sólo estaban disponibles unos pocos gestos como: movimiento circular del dedo, pulsación de una tecla en el aire, barrido de un dedo en el aire y tap o toque de la pantalla en el aire con un dedo. Posteriormente se agregaron otros.

Como desarrollador, debo confesar que la precisión era bastante decepcionante. Pronto me encontré con dificultades inesperadas no relacionadas con la adaptación del game play. Una tarea cotidiana y simple como pulsar un botón del menú se convirtió en una tarea difícil y frustrante. Llegué a pensar que lo que veía en los videos de Leap Motion eran versiones altamente glorificadas por el marketing y los programas de edición. Sin embargo, decidí seguir adelante con el proyecto porque me gustaba la idea de controlar mi videojuego con las manos, esa libertad es imposible de reproducir con ninguno de los periféricos convencionales de la PC.

Estoy muy feliz y orgulloso de crear mi primer videojuego para leap Motion. Espero que más desarrolladores latinoamericanos se animen a trabajar con esta nueva tecnología.

3.2 EMPEZANDO A CREAR APLICACIONES CON LEAP MOTION SDK Y UNITY3D (VERSIÓN GRATUITA)

Comienzo con el primero de una serie de tutoriales básicos para que puedas sacar de la caja tu Leap Motion y comenzar a programar tus primeras líneas de código. Este tutorial está realizado para aquellos que no posean la versión de pago de Unity3D.

El material contenido en este tutorial está extraído de la página oficial de Leap Motion y de sus documentos de ayuda para desarrolladores.

La licencia standard de Unity (Free) no soporta plugins, pero gracias a la comunidad de desarrolladores de Leap Motion es posible utilizar las librerías del Leap con esta versión de Unity3D. Esto requiere una serie de pasos manuales que hay que seguir:

Para hacer funcionar un proyecto con las librerías del Leap debes ponerlas en un lugar donde puedan ser encontradas en runtime. Este lugar es distinto dependiendo de donde estés ejecutando el proyecto, en el editor de Unity o en el ejecutable después de exportarlo.

Dado que la versión estándar de Unity no es compatible con plugins, la clave está en poner las librerías en un lugar donde el sistema operativo puede encontrarlas.

COLOCACIÓN DE LAS LIBRERÍAS DURANTE EL DESARROLLO

Mientras estemos en desarrollo podemos colocar las librerías de .NET3.5 en la carpeta Assets de nuestro proyecto Unity y las librerías del Leap y LeapCSharp en la raíz del proyecto.

WINDOWS

```
Ejemplo/
   Assets/
         LeapCSharp.NET3.5.dll
         ejemplo.unity
           Scripts/
                    LeapControl.cs
   Library/
   ProjectSettings/
          Leap.dll
          LeapCSharp.dll
           msvcp100.dll
         msvcr100.dll
```

En Windows, usar solo las librerías de 32-bit en la carpeta x86 con el editor.

OS X

```
Ejemplo/
   Assets/
      LeapCSharp.NET3.5.dll
      ejemplo.unity
      Scripts/
         LeapControl.cs
   Library/
   ProjectSettings/
   libLeap.dylib
   libLeapCSharp.dylib
```

En Mac, no usar el archivo LeapCSharp.bundle desde el Leap SDK. Las librerías deben de copiarse por separado.

COLOCACIÓN DE LAS LIBRERÍAS DESPUÉS DE EXPORTAR

Después de exportar nuestro proyecto Unity, también debemos copiar las librerías del Leap y sus dependencias en la carpeta exportada junto a la aplicación. Por ejemplo, si tú exportas tu proyecto a una carpeta llamada ejemplo, el contenido de la carpeta final quedará de la siguiente forma.

WINDOWS

```
Ejemploexport/
    Leap.dll
    LeapCSharp.dll
    msvcp100.dll
    msvcr100.dll
    Ejemplo_Data/
    Ejemplo.exe
```

Hay que asegurarse de que estamos usando las librerías de la carpeta x86 en el SDK para Windows 32-bit y las librerías de la carpeta x64 para los Windows de 64-bit.

OS X:

```
EjemploExport/
    libLeap.dylib
    libLeapCSharp.dylib
    Ejemplo.app
```

Ahora podemos hacer un instalador para distribuir nuestra aplicación de la misma forma que podríamos hacerlo para una aplicación que no estuviera preparada para usar con el Leap.

COLOCACIÓN DE LAS LIBRERÍAS DEL LEAP DENTRO DEL PAQUETE DE LA APLICACIÓN EN MAC

Es conveniente colocar todas las dependencias dentro del paquete de la aplicación en el Mac. Para ello, debes identificar la ruta donde se encuentran las librerías. Puedes utilizar la utilidad **install_name_tool** para agregar la ruta a tu binario y copiar las librerías en la carpeta del Framework en el paquete de la aplicación. Por ejemplo, el paquete para la supuesta aplicación **Ejemplo** quedaría de la siguiente manera después de exportarla con Unity:

```
Ejemplo.app/
   Data/
   Frameworks/
      MonoEmbedRuntime/
   Info.plist
   MacOS/
     Ejemplo
   PkgInfo
   Resources/
```

Primero, debes agregar una ruta de acceso relativa al binario de la aplicación utilizando install_name_tool. Desde la carpeta de exportación, ejecuta el siguiente comando en una ventana del Terminal:

```
install_name_tool -add_rpath @executable_path/../Frameworks
UnicornSlam.app/Contents/MacOS/UnicornSlam
```

A continuación, copia libLeap.dylib y libLeapCSharp.dylib en la carpeta de los Frameworks dentro del paquete. Después de todo esto, el paquete de la app debería quedar así:

```
Ejemplo.app/
   Data/
   Frameworks/
      libLeap.dylib
      libLeapCSharp.dylib
      MonoEmbedRuntime/
   Info.plist
   MacOS/
         Ejemplo
   PkgInfo
   Resources/
```

IV. CONSIDERACIONES FINALES

4.1 CONCLUSIONES

- Con el desarrollo de este material se demostró lo fácil que es el funcionamiento del dispositivo LEAP MOTION, este facilita la interacción entre persona y ordenador mediante gestos intuitivos que se realizan en el aire con nuestras manos, ofreciendo infinitas posibilidades al usuario.
- La aplicación desarrollada también se muestra como una diferencia en la forma de optar por este tipo de tecnología, pues comparado con otros sistemas, teóricamente es más eficiente, más flexible y fácil de usar.
- A pesar de que el dispositivo está al alcance de todos por su bajo precio en el mercado, todavía no ha recibido el suficiente apoyo por parte de los desarrolladores, y por lo tanto a día de hoy ofrece pocas aplicaciones. Sin embargo, creo que este dispositivo innovador tiene muchas posibilidades de desarrollo y un futuro prometedor. Además, cabe destacar que ya existen ordenadores en el mercado que integran dicha tecnología.

4.2 TRABAJOS FUTUROS

Una propuesta inicial para el trabajo futuro finalizaría el despliegue de historias planteadas y evaluar la usabilidad de la aplicación. La implementación de estas características siguen el mismo procedimiento que ya se utiliza, requiriendo solamente una definición de los prototipos de los nuevos gestos, y la evaluación de la propuesta sobre el nuevo paradigma de la interacción con los sistemas informáticos sería considerado desde el punto de vista del usuario.

La propuesta para evaluar la facilidad de uso de la aplicación podría basarse la métrica: la ciencia de uso, facilidad de aprendizaje y satisfacción del usuario en su uso.

REFERENCIAS BIBLIOGRAFICAS

1) ALVARENGA, Matheus L., Correa, Diogo S. e Osório, Fernando S. 2012. Redes Neurais Artificiais aplicadas no Reconhecimento de Gestos usando o Kinect. Universidade de São Paulo. 2012. p. 10.

2) ASUS. 2013 Xtion PRO developer solution to make motion-sensing applications and games [Online] 2015. http://www.asus.com/Multimedia/Xtion_PRO_LIVE/.

3) AZIMI, Mehran. 2015. Skeletal Joint Smoothing White Paper [Online] 2015 http://msdn.microsoft.com/en-us/library/jj131429.aspx

4) CATUHE, David. 2012. Programming with the Kinect for Windows Sofware Develpmento Kit. Redmond, Washington. Microsoft Press, 2012. p. 210. ISBN: 978-0-7356-6681-8.

5) GARBIN, Sander M. 2010. Estudo da evolução das interfaces homem-computador.
Universidade de São Paulo, São Carlos.

6) GHIROTTI, Silvia E. y MORIMOTO, Carlos H. 2009. Un sistema de interacción basado en gestos de la mano en tres dimensiones a los entornos virtuales. Departamento de Ciencias de la Computación - IME / USP

7) MEDEIROS, Anna Carolina S. 2012. Interacción Natural basada en gestos como Control de interfaz para los modelos tridimensionales. Universidad Federal Paraíba, 2012. 63 p

8) MOTION CONTROL, «Leap Motion,» 2014. [En línea]. Available: https://www.leapmotion.com/. [Último acceso: 13 -12- 2015].

9) MOTION CONTROL, «Leap Motion Airspace,» [En línea]. Available: https://airspace.leapmotion.com/categories/all. [Último acceso: 13- 12- 2015].

10) NORMAM, Donald. 1986. A. User Centered System Design: New Perspectives on Human-Computer Interaction.

11) PREECE, Jennifer, ROGERS, Yvonne, SHARP, Helen. Diseño de Interacción. Más allá de la interacción casa - ordenador. Editorial Bookman. Porto Alegre - RS, 2005. 348p.

12) RIBEIRO, H. L. 2006. Reconhecimento de gestos usando segmentação de imagens dinâmicas de mãos baseada no modelo de mistura de Gaussianas e cor de pele. 2006. 144p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, SP.

13) S. MITRA E T. ACHARYA. 2007. Reconhecimento de Gestos: Uma Pesquisa. Sistemas, Homem, e Cibernético, Transcrição IEEE no, 37(3):311-324.

14) SBC. 2013. Sociedade Brasileira de Computação. Interação Humano Computador.[Online] 2015. http://www.sbc.org.br/index.php?option=com_content&view=category&layout=blog&id=45&Itemid=66.

15) TOGORES, Thiago Andrade. 2011. Vitruvius - Um Reconhecedor de Gestos para o Kinect. Universidade de São Paulo. 2011. p. 42.

ANEXO

DESARROLLO DE APLICACION EN UNITY 3D Y C#

TITULO: CONDUCE CON LEAP

Figura 20. Pantalla de inicio de la aplicación

Paso 1 Crear un nuevo proyecto.

- **Nombre de la Aplicación " Conduce con Leap"**

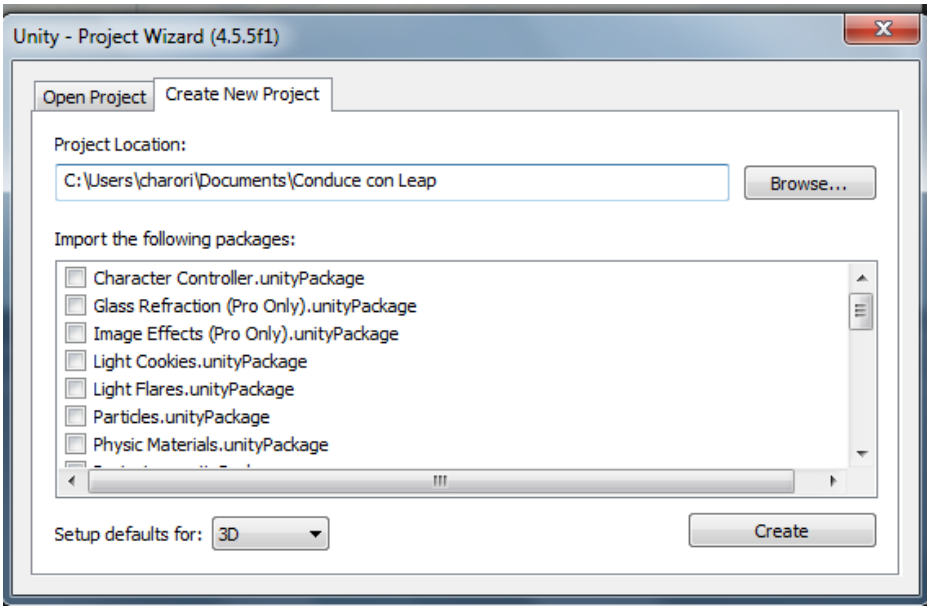


Figura 21 Creación del proyecto

Paso 2 Asegúrese de que la escena completa está funcionando correctamente

Figura 22 Plataforma de desarrollo de Unity 3D

Paso 3 Crear una carpeta Plugins en el directorio activo

Paso 4 Copie los archivos de Salta de UnityAssets en el SDK a su directorio Directorio y raíz Plugins

- Si tiene la unidad Pro, puede copiar los 3 archivos (Windows) de salto de movimiento a su director Plugins recién creada.
- Si tiene la Unidad gratuito sólo, entonces usted tiene que copiar Leap.dll y LeapCSharp.dll al directorio raíz de su juego, y LeapCSharp.NET3.5.dll a su directorio de plugins.

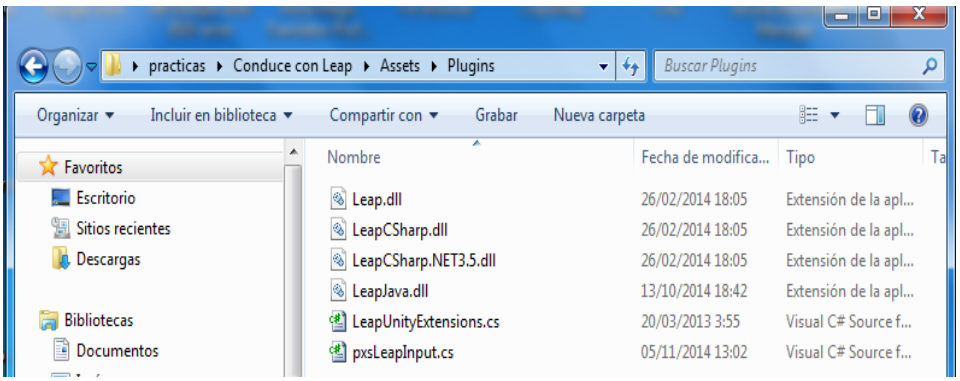


Figura 23 Creación de la carpeta Plugins

Paso 5 Crear clase LeapUnityExtensions.cs en la carpeta Plugins

```
using UnityEngine;
using System.Collections;
using Leap;

namespace Leap {


	public static class UnityVectorExtension
	{
		public static Vector3 InputScale = new Vector3(0.02f, 0.02f, 0.02f);
		public static Vector3 InputOffset = new Vector3(0,0,0);

		//para Direcciones
		public static Vector3 ToUnity(this Vector lv)
		{
			return FlippedZ(lv);
		}
		//Para Aceleracion y Velocidad
		public static Vector3 ToUnityScaled(this Vector lv)
		{
			return Scaled(FlippedZ( lv ));
		}
		//Para Posiciones
		public static Vector3 ToUnityTranslated(this Vector lv)
		{
			return Offset(Scaled(FlippedZ( lv )));
		}

		private static Vector3 FlippedZ( Vector v ) { return new Vector3( v.x, v.y, -v.z ); }
		private static Vector3 Scaled( Vector3 v ) { return new Vector3( v.x * InputScale.x,

                                                                 v.y * InputScale.y,

                                                                 v.z * InputScale.z ); }
		private static Vector3 Offset( Vector3 v ) { return v + InputOffset; }
	}
}
```

Paso 6 Crear clase pxsLeapInput.cs en la carpeta Plugins

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using Leap;

/// <summary>

/// </summary>
public static class pxsLeapInput
{
	private enum HandID : int
	{
		Primary		= 0,
		Secondary	= 1
	};

		static Leap.Controller		m_controller	= null;
	static Leap.Frame			m_Frame			= null;
```

```
static Leap.Hand                m_Hand                  = null;
static int                      m_FingersCount          = 0;
static string m_Errors                          = "";


static pxsLeapInput()
{
        try
        {
                m_controller       = new Leap.Controller();
        }
        catch
        {
                m_Errors = m_Errors + '\r' + '\n'  + "Controller could not be created";
        }
}

public static Leap.Frame Frame
{
        get
        {
                Update(); return m_Frame;
        }
}
public static int FingersCount
{
        get
        {
                Update();
                return m_FingersCount;
        }
}

public static Leap.Hand Hand
{

        get
        {
                Update();
                return m_Hand;
        }
}

public static string Errors
{
        get { return m_Errors; }
}

public static void Update()
{
        m_Hand = null;
        m_Frame = null;
        m_FingersCount = 0;
        if( m_controller != null )
        {
                m_Frame          = m_controller.Frame();
                if (m_Frame != null)
                {
                        if (m_Frame.Hands.Count > 0)
                        {
```

```
                                        m_Hand = m_Frame.Hands[0];
                                }

                                if (m_Frame.Fingers.Count > 0)
                                {
                                        m_FingersCount = m_Frame.Fingers.Count;
                                }
                        }
                }
        }

        // returns the hand axis scaled from -1 to +1
        public static float GetHandAxis(string axisName)
        {
                float ret = GetHandAxisPrivate(axisName, true);
                return ret;
        }

        public static float GetHandAxisStep(string axisName)
        {
                // Assume dead zone of -0.5 to 0.5
                // always return -1, 0 or +1
                float ret = GetHandAxisPrivate(axisName, true);
                if (ret < -0.5F)
                {
                        ret = -1.0F;
                }
                else if (ret > 0.5F)
                {
                        ret = 1.0F;
                }
                else
                {
                        ret = 0;
                }
                return ret;
        }


        public static float GetHandAxisRaw(string axisName)
        {
                float ret = GetHandAxisPrivate(axisName, false);
                return ret;
        }

        public static bool GetHandGesture(string gestureName)
        {
                // Call Update so you can get the latest frame and hand
                Update();
                bool ret = false;

                Vector3 PalmNormal = new Vector3(0,0,0);
                if (m_Hand != null)
                {

                        PalmNormal = m_Hand.PalmNormal.ToUnity();


                        switch (gestureName)
                        {
```

```
                        case "Fire1":
                                ret =  m_FingersCount > 2;
                                break;
                        case "Fire2":
                                ret =  (PalmNormal.x > 0.5) ;
                                break;
                        case "RotationFire2":
                                ret =  (PalmNormal.x < -0.25) ;
                                break;
                        default:
                                break;
                        }
                }
                return ret;
        }

        private static float GetHandAxisPrivate(string axisName, bool scaled)
        {
                // Call Update so you can get the latest frame and hand
                Update();
                float ret = 0.0F;
                if (m_Hand != null)
                {
                        Vector3 PalmPosition = new Vector3(0,0,0);
                        Vector3 PalmNormal = new Vector3(0,0,0);
                        Vector3 PalmDirection = new Vector3(0,0,0);
                        if (scaled == true)
                        {
                                PalmPosition = m_Hand.PalmPosition.ToUnityTranslated();
                                PalmNormal = m_Hand.PalmNormal.ToUnity();

                                PalmDirection = m_Hand.Direction.ToUnity();
                        }
                        else
                        {
                                PalmPosition = m_Hand.PalmPosition.ToUnity();
                                PalmNormal = m_Hand.PalmPosition.ToUnity();
                                PalmDirection = m_Hand.Direction.ToUnity();
                        }

                        switch (axisName)
                        {
                        case "Horizontal":
                                ret =  PalmPosition.x ;
                                break;
                        case "Vertical":
                                ret =  PalmPosition.y ;
                                break;
                        case "Mouse X":
                                // Uncomment below if using Rotation for Mouse X.
                                // ret = ret = -2 * PalmNormal.x ;
                                // Uncomment below if using Direction for Mouse X.
                                ret =  4 * (PalmDirection.x ) + 1.00F ;
                                // m_Errors = "ret Mouse.x = " + ret.ToString();
                                break;
                        case "Mouse Y":
                                // use z axis as this is the most promounced change when TITLTing a
hand
                                ret =  2 * PalmNormal.z ;
                                // m_Errors = "ret Mouse.Y = " + ret.ToString();
```

```
                                        break;

                                case "Depth":
                                        ret =  PalmPosition.z;
                                        break;
                                case "Rotation":
                                        ret = -2 * PalmNormal.x ;
                                        break;
                                case "Tilt":
                                        ret = PalmNormal.z ;
                                        break;
                                case "HorizontalDirection":
                                        ret = PalmDirection.x ;
                                        break;
                                case "VericalDirection":
                                        ret = PalmDirection.y ;
                                        break;
                                default:
                                        break;
                                }

                                if (scaled == true)
                                        {
                                                if (ret > 1) {ret = 1;}
                                                if (ret < -1) {ret = -1;}
                                        }

                        }
                        else
                        {
                                // Hand is Null, so return the standard axis
                                switch (axisName)
                                {
                                case "Depth":
                                        //depth for leap = vertical axis
                                        ret = Input.GetAxis("Vertical") ;
                                        break;

//                              case "Rotation":
//                                      ret = Input.GetAxis("Horizontal");
//                                      break;

                                default:
                                        // return the axis name from input if hand is null and the special cases
above do not apply
                                        ret = Input.GetAxis(axisName);

                                        break;
                                }

                        }
                        return ret;


}
}
```

Paso 7 Crear car.js script para permitir o Salto de entrada

```
public var LeapEnabled: boolean=false;
```

```
private var wheelRadius : float = 0.4;
var suspensionRange : float = 0.1;
var suspensionDamper : float = 50;
var suspensionSpringFront : float = 18500;
var suspensionSpringRear : float = 9000;

public var brakeLights : Material;

var dragMultiplier : Vector3 = new Vector3(2, 5, 1);

var throttle : float = 0;
private var steer : float = 0;
private var handbrake : boolean = false;

var centerOfMass : Transform;

var frontWheels : Transform[];
var rearWheels : Transform[];

private var wheels : Wheel[];
wheels = new Wheel[frontWheels.Length + rearWheels.Length];

private var wfc : WheelFrictionCurve;

var topSpeed : float = 160;
var numberOfGears : int = 5;

var maximumTurn : int = 15;
var minimumTurn : int = 10;

var resetTime : float = 5.0;
private var resetTimer : float = 0.0;

private var engineForceValues : float[];
private var gearSpeeds : float[];

private var currentGear : int;
private var currentEnginePower : float = 0.0;

private var handbrakeXDragFactor : float = 0.5;
private var initialDragMultiplierX : float = 10.0;
private var handbrakeTime : float = 0.0;
private var handbrakeTimer : float = 1.0;

private var skidmarks : Skidmarks = null;
private var skidSmoke : ParticleEmitter = null;
var skidmarkTime : float[];

private var sound : SoundController = null;
sound = transform.GetComponent(SoundController);

private var canSteer : boolean;
private var canDrive : boolean;

class Wheel
{
        var collider : WheelCollider;
        var wheelGraphic : Transform;
        var tireGraphic : Transform;
        var driveWheel : boolean = false;
```

```
	var steerWheel : boolean = false;
	var lastSkidmark : int = -1;
	var lastEmitPosition : Vector3 = Vector3.zero;
	var lastEmitTime : float = Time.time;
	var wheelVelo : Vector3 = Vector3.zero;
	var groundSpeed : Vector3 = Vector3.zero;
}

function Start()
{
	// Measuring 1 - 60
	accelerationTimer = Time.time;

	SetupWheelColliders();

	SetupCenterOfMass();

	topSpeed = Convert_Miles_Per_Hour_To_Meters_Per_Second(topSpeed);

	SetupGears();

	SetUpSkidmarks();

	initialDragMultiplierX = dragMultiplier.x;
}

function Update()
{
	var relativeVelocity : Vector3 = transform.InverseTransformDirection(rigidbody.velocity);

	GetInput();

	Check_If_Car_Is_Flipped();

	UpdateWheelGraphics(relativeVelocity);

	UpdateGear(relativeVelocity);
}

function FixedUpdate()
{
	// The rigidbody velocity is always given in world space, but in order to work in local space of
the car model we need to transform it first.
	var relativeVelocity : Vector3 = transform.InverseTransformDirection(rigidbody.velocity);

	CalculateState();

	UpdateFriction(relativeVelocity);

	UpdateDrag(relativeVelocity);

	CalculateEnginePower(relativeVelocity);

	ApplyThrottle(canDrive, relativeVelocity);

	ApplySteering(canSteer, relativeVelocity);
}

/*****************************************************/
/* Functions called from Start()		*/
```

```
/*********************************************************/

function SetupWheelColliders()
{
	SetupWheelFrictionCurve();

	var wheelCount : int = 0;

	for (var t : Transform in frontWheels)
	{
		wheels[wheelCount] = SetupWheel(t, true);
		wheelCount++;
	}

	for (var t : Transform in rearWheels)
	{
		wheels[wheelCount] = SetupWheel(t, false);
		wheelCount++;
	}
}

function SetupWheelFrictionCurve()
{
	wfc = new WheelFrictionCurve();
	wfc.extremumSlip = 1;
	wfc.extremumValue = 50;
	wfc.asymptoteSlip = 2;
	wfc.asymptoteValue = 25;
	wfc.stiffness = 1;
}

function SetupWheel(wheelTransform : Transform, isFrontWheel : boolean)
{
	var go : GameObject = new GameObject(wheelTransform.name + " Collider");
	go.transform.position = wheelTransform.position;
	go.transform.parent = transform;
	go.transform.rotation = wheelTransform.rotation;

	var wc : WheelCollider = go.AddComponent(typeof(WheelCollider)) as WheelCollider;
	wc.suspensionDistance = suspensionRange;
	var js : JointSpring = wc.suspensionSpring;

	if (isFrontWheel)
		js.spring = suspensionSpringFront;
	else
		js.spring = suspensionSpringRear;

	js.damper = suspensionDamper;
	wc.suspensionSpring = js;

	wheel = new Wheel();
	wheel.collider = wc;
	wc.sidewaysFriction = wfc;
	wheel.wheelGraphic = wheelTransform;
	wheel.tireGraphic = wheelTransform.GetComponentsInChildren(Transform)[1];

	wheelRadius = wheel.tireGraphic.renderer.bounds.size.y / 2;
	wheel.collider.radius = wheelRadius;

	if (isFrontWheel)
```

```
		{
			wheel.steerWheel = true;

			go = new GameObject(wheelTransform.name + " Steer Column");
			go.transform.position = wheelTransform.position;
			go.transform.rotation = wheelTransform.rotation;
			go.transform.parent = transform;
			wheelTransform.parent = go.transform;
		}
		else
			wheel.driveWheel = true;

		return wheel;
}

function SetupCenterOfMass()
{
		if(centerOfMass != null)
			rigidbody.centerOfMass = centerOfMass.localPosition;
}

function SetupGears()
{
		engineForceValues = new float[numberOfGears];
		gearSpeeds = new float[numberOfGears];

		var tempTopSpeed : float = topSpeed;

		for(var i = 0; i < numberOfGears; i++)
		{
			if(i > 0)
				gearSpeeds[i] = tempTopSpeed / 4 + gearSpeeds[i-1];
			else
				gearSpeeds[i] = tempTopSpeed / 4;

			tempTopSpeed -= tempTopSpeed / 4;
		}

		var engineFactor : float = topSpeed / gearSpeeds[gearSpeeds.Length - 1];

		for(i = 0; i < numberOfGears; i++)
		{
			var maxLinearDrag : float = gearSpeeds[i] * gearSpeeds[i];// * dragMultiplier.z;
			engineForceValues[i] = maxLinearDrag * engineFactor;
		}
}

function SetUpSkidmarks()
{
		if(FindObjectOfType(Skidmarks))
		{
			skidmarks = FindObjectOfType(Skidmarks);
			skidSmoke = skidmarks.GetComponentInChildren(ParticleEmitter);
		}
		else
			Debug.Log("No skidmarks object found. Skidmarks will not be drawn");

		skidmarkTime = new float[4];
		for (var f : float in skidmarkTime)
			f = 0.0;
```

```
}

/*****************************************************/
/* Functions called from Update()                    */
/*****************************************************/

function GetInput()
{
        if (LeapEnabled == true)
{
 throttle = pxsLeapInput.GetHandAxis("Depth");
 print("Throttle: " + throttle.ToString());
 steer = pxsLeapInput.GetHandAxis("Rotation");
 print("Steer: " + steer.ToString());
}
else
{
 throttle = Input.GetAxis("Vertical");
 steer = Input.GetAxis("Horizontal");
}
}

function CheckHandbrake()
{
        if(Input.GetKey("space"))
        {
                if(!handbrake)
                {
                        handbrake = true;
                        handbrakeTime = Time.time;
                        dragMultiplier.x = initialDragMultiplierX * handbrakeXDragFactor;
                }
        }
        else if(handbrake)
        {
                handbrake = false;
                StartCoroutine(StopHandbraking(Mathf.Min(5, Time.time - handbrakeTime)));
        }
}

function StopHandbraking(seconds : float)
{
        var diff : float = initialDragMultiplierX - dragMultiplier.x;
        handbrakeTimer = 1;

        // Get the x value of the dragMultiplier back to its initial value in the specified time.
        while(dragMultiplier.x < initialDragMultiplierX && !handbrake)
        {
                dragMultiplier.x += diff * (Time.deltaTime / seconds);
                handbrakeTimer -= Time.deltaTime / seconds;
                yield;
        }

        dragMultiplier.x = initialDragMultiplierX;
        handbrakeTimer = 0;
}

function Check_If_Car_Is_Flipped()
{
        if(transform.localEulerAngles.z > 80 && transform.localEulerAngles.z < 280)
```

```
			resetTimer += Time.deltaTime;
		else
			resetTimer = 0;

		if(resetTimer > resetTime)
			FlipCar();
}

function FlipCar()
{
		transform.rotation = Quaternion.LookRotation(transform.forward);
		transform.position += Vector3.up * 0.5;
		rigidbody.velocity = Vector3.zero;
		rigidbody.angularVelocity = Vector3.zero;
		resetTimer = 0;
		currentEnginePower = 0;
}

var wheelCount : float;
function UpdateWheelGraphics(relativeVelocity : Vector3)
{
		wheelCount = -1;

		for(var w : Wheel in wheels)
		{
			wheelCount++;
			var wheel : WheelCollider = w.collider;
			var wh : WheelHit = new WheelHit();

			// First we get the velocity at the point where the wheel meets the ground, if the wheel is
touching the ground
			if(wheel.GetGroundHit(wh))
			{
				w.wheelGraphic.localPosition = wheel.transform.up * (wheelRadius +
wheel.transform.InverseTransformPoint(wh.point).y);
				w.wheelVelo = rigidbody.GetPointVelocity(wh.point);
				w.groundSpeed = w.wheelGraphic.InverseTransformDirection(w.wheelVelo);

				// Code to handle skidmark drawing. Not covered in the tutorial
				if(skidmarks)
				{
					if(skidmarkTime[wheelCount] < 0.02 && w.lastSkidmark != -1)
					{
						skidmarkTime[wheelCount] += Time.deltaTime;
					}
					else
					{
						var dt : float = skidmarkTime[wheelCount] == 0.0 ?
Time.deltaTime : skidmarkTime[wheelCount];
						skidmarkTime[wheelCount] = 0.0;

						var handbrakeSkidding : float = handbrake &&
w.driveWheel ? w.wheelVelo.magnitude * 0.3 : 0;
						var skidGroundSpeed = Mathf.Abs(w.groundSpeed.x) - 15;
						if(skidGroundSpeed > 0 || handbrakeSkidding > 0)
						{
							var staticVel : Vector3 =
transform.TransformDirection(skidSmoke.localVelocity) + skidSmoke.worldVelocity;
							if(w.lastSkidmark != -1)
							{
```

```
                                                                var emission : float =
UnityEngine.Random.Range(skidSmoke.minEmission, skidSmoke.maxEmission);
                                                                var lastParticleCount : float =
w.lastEmitTime * emission;
                                                                var currentParticleCount : float = Time.time
* emission;
                                                                var noOfParticles : int =
Mathf.CeilToInt(currentParticleCount) - Mathf.CeilToInt(lastParticleCount);
                                                                var lastParticle : int =
Mathf.CeilToInt(lastParticleCount);

                                                                for(var i = 0; i <= noOfParticles; i++)
                                                                {
                                                                        var particleTime : float =
Mathf.InverseLerp(lastParticleCount, currentParticleCount, lastParticle + i);
                                                                        skidSmoke.Emit(
        Vector3.Lerp(w.lastEmitPosition, wh.point, particleTime) + new Vector3(Random.Range(-0.1,
0.1), Random.Range(-0.1, 0.1), Random.Range(-0.1, 0.1)), staticVel + (w.wheelVelo * 0.05),
Random.Range(skidSmoke.minSize, skidSmoke.maxSize) * Mathf.Clamp(skidGroundSpeed * 0.1,0.5,1),
Random.Range(skidSmoke.minEnergy, skidSmoke.maxEnergy), Color.white);
                                                                }
                                                        }
                                                        else
                                                        {
                                                                skidSmoke.Emit( wh.point + new
Vector3(Random.Range(-0.1, 0.1), Random.Range(-0.1, 0.1), Random.Range(-0.1, 0.1)), staticVel +
(w.wheelVelo * 0.05), Random.Range(skidSmoke.minSize, skidSmoke.maxSize) *
Mathf.Clamp(skidGroundSpeed * 0.1,0.5,1), Random.Range(skidSmoke.minEnergy,
skidSmoke.maxEnergy), Color.white);
                                                        }

                                                        w.lastEmitPosition = wh.point;
                                                        w.lastEmitTime = Time.time;

                                                        w.lastSkidmark =
skidmarks.AddSkidMark(wh.point + rigidbody.velocity * dt, wh.normal, (skidGroundSpeed * 0.1 +
handbrakeSkidding) * Mathf.Clamp01(wh.force / wheel.suspensionSpring.spring), w.lastSkidmark);
                                                        sound.Skid(true, Mathf.Clamp01(skidGroundSpeed
* 0.1));
                                                }
                                                else
                                                {
                                                        w.lastSkidmark = -1;
                                                        sound.Skid(false, 0);
                                                }
                                        }
                                }
                        }
                        else
                        {
                                // If the wheel is not touching the ground we set the position of the wheel
graphics to
                                // the wheel's transform position + the range of the suspension.
                                w.wheelGraphic.position = wheel.transform.position + (-wheel.transform.up *
suspensionRange);
                                if(w.steerWheel)
                                        w.wheelVelo *= 0.9;
                                else
                                        w.wheelVelo *= 0.9 * (1 - throttle);
```

```
                        if(skidmarks)
                        {
                                w.lastSkidmark = -1;
                                sound.Skid(false, 0);
                        }
                }
                // If the wheel is a steer wheel we apply two rotations:
                // *Rotation around the Steer Column (visualizes the steer direction)
                // *Rotation that visualizes the speed
                if(w.steerWheel)
                {
                        var ea : Vector3 = w.wheelGraphic.parent.localEulerAngles;
                        ea.y = steer * maximumTurn;
                        w.wheelGraphic.parent.localEulerAngles = ea;
                        w.tireGraphic.Rotate(Vector3.right * (w.groundSpeed.z / wheelRadius) *
Time.deltaTime * Mathf.Rad2Deg);
                }
                else if(!handbrake && w.driveWheel)
                {
                        // If the wheel is a drive wheel it only gets the rotation that visualizes speed.
                        // If we are hand braking we don't rotate it.
                        w.tireGraphic.Rotate(Vector3.right * (w.groundSpeed.z / wheelRadius) *
Time.deltaTime * Mathf.Rad2Deg);
                }
        }
}

function UpdateGear(relativeVelocity : Vector3)
{
        currentGear = 0;
        for(var i = 0; i < numberOfGears - 1; i++)
        {
                if(relativeVelocity.z > gearSpeeds[i])
                        currentGear = i + 1;
        }
}

/*****************************************************/
/* Functions called from FixedUpdate()          */
/*****************************************************/

function UpdateDrag(relativeVelocity : Vector3)
{
        var relativeDrag : Vector3 = new Vector3(   -relativeVelocity.x * Mathf.Abs(relativeVelocity.x),

        -relativeVelocity.y * Mathf.Abs(relativeVelocity.y),

        -relativeVelocity.z * Mathf.Abs(relativeVelocity.z) );

        var drag = Vector3.Scale(dragMultiplier, relativeDrag);

        if(initialDragMultiplierX > dragMultiplier.x) // Handbrake code
        {
                drag.x /= (relativeVelocity.magnitude / (topSpeed / ( 1 + 2 * handbrakeXDragFactor ) )
);
                drag.z *= (1 + Mathf.Abs(Vector3.Dot(rigidbody.velocity.normalized,
transform.forward)));
                drag += rigidbody.velocity * Mathf.Clamp01(rigidbody.velocity.magnitude / topSpeed);
        }
        else // No handbrake
```

```
		{
			drag.x *= topSpeed / relativeVelocity.magnitude;
		}

		if(Mathf.Abs(relativeVelocity.x) < 5 && !handbrake)
			drag.x = -relativeVelocity.x * dragMultiplier.x;


		rigidbody.AddForce(transform.TransformDirection(drag) * rigidbody.mass * Time.deltaTime);
}

function UpdateFriction(relativeVelocity : Vector3)
{
		var sqrVel : float = relativeVelocity.x * relativeVelocity.x;

		// Add extra sideways friction based on the car's turning velocity to avoid slipping
		wfc.extremumValue = Mathf.Clamp(300 - sqrVel, 0, 300);
		wfc.asymptoteValue = Mathf.Clamp(150 - (sqrVel / 2), 0, 150);

		for(var w : Wheel in wheels)
		{
			w.collider.sidewaysFriction = wfc;
			w.collider.forwardFriction = wfc;
		}
}

function CalculateEnginePower(relativeVelocity : Vector3)
{
		if(throttle == 0)
		{
			currentEnginePower -= Time.deltaTime * 200;
		}
		else if( HaveTheSameSign(relativeVelocity.z, throttle) )
		{
			normPower = (currentEnginePower / engineForceValues[engineForceValues.Length -
1]) * 2;
			currentEnginePower += Time.deltaTime * 200 * EvaluateNormPower(normPower);
		}
		else
		{
			currentEnginePower -= Time.deltaTime * 300;
		}

		if(currentGear == 0)
			currentEnginePower = Mathf.Clamp(currentEnginePower, 0, engineForceValues[0]);
		else
			currentEnginePower = Mathf.Clamp(currentEnginePower,
engineForceValues[currentGear - 1], engineForceValues[currentGear]);
}

function CalculateState()
{
		canDrive = false;
		canSteer = false;

		for(var w : Wheel in wheels)
		{
			if(w.collider.isGrounded)
			{
				if(w.steerWheel)
```

```
                                canSteer = true;
                        if(w.driveWheel)
                                canDrive = true;
                }
        }
}

function ApplyThrottle(canDrive : boolean, relativeVelocity : Vector3)
{
        if(canDrive)
        {
                var throttleForce : float = 0;
                var brakeForce : float = 0;

                if (HaveTheSameSign(relativeVelocity.z, throttle))
                {
                        if (!handbrake)
                                throttleForce = Mathf.Sign(throttle) * currentEnginePower *
rigidbody.mass;
                }
                else
                        brakeForce = Mathf.Sign(throttle) * engineForceValues[0] * rigidbody.mass;

                rigidbody.AddForce(transform.forward * Time.deltaTime * (throttleForce +
brakeForce));
        }
}

function ApplySteering(canSteer : boolean, relativeVelocity : Vector3)
{
        if(canSteer)
        {
                var turnRadius : float = 3.0 / Mathf.Sin((90 - (steer * 30)) * Mathf.Deg2Rad);
                var minMaxTurn : float = EvaluateSpeedToTurn(rigidbody.velocity.magnitude);
                var turnSpeed : float = Mathf.Clamp(relativeVelocity.z / turnRadius, -minMaxTurn /
10, minMaxTurn / 10);

                transform.RotateAround(  transform.position + transform.right * turnRadius * steer,
                                                                transform.up,
                                                                turnSpeed * Mathf.Rad2Deg *
Time.deltaTime * steer);

                var debugStartPoint = transform.position + transform.right * turnRadius * steer;
                var debugEndPoint = debugStartPoint + Vector3.up * 5;

                Debug.DrawLine(debugStartPoint, debugEndPoint, Color.red);

                if(initialDragMultiplierX > dragMultiplier.x) // Handbrake
                {
                        var rotationDirection : float = Mathf.Sign(steer); // rotationDirection is -1 or 1
by default, depending on steering
                        if(steer == 0)
                        {
                                if(rigidbody.angularVelocity.y < 1) // If we are not steering and we are
handbraking and not rotating fast, we apply a random rotationDirection
                                        rotationDirection = Random.Range(-1.0, 1.0);
                                else
                                        rotationDirection = rigidbody.angularVelocity.y; // If we are
rotating fast we are applying that rotation to the car
                        }
```

```
                        // -- Finally we apply this rotation around a point between the cars front
wheels.
                        transform.RotateAround( transform.TransformPoint( (
        frontWheels[0].localPosition + frontWheels[1].localPosition) * 0.5),

                                transform.up,

                                rigidbody.velocity.magnitude * Mathf.Clamp01(1 -
rigidbody.velocity.magnitude / topSpeed) * rotationDirection * Time.deltaTime * 2);
                }
        }
}

/**************************************************/
/*          Utility Functions          */
/**************************************************/

function Convert_Miles_Per_Hour_To_Meters_Per_Second(value : float) : float
{
        return value * 0.44704;
}

function Convert_Meters_Per_Second_To_Miles_Per_Hour(value : float) : float
{
        return value * 2.23693629;
}

function HaveTheSameSign(first : float, second : float) : boolean
{
        if (Mathf.Sign(first) == Mathf.Sign(second))
                return true;
        else
                return false;
}

function EvaluateSpeedToTurn(speed : float)
{
        if(speed > topSpeed / 2)
                return minimumTurn;

        var speedIndex : float = 1 - (speed / (topSpeed / 2));
        return minimumTurn + speedIndex * (maximumTurn - minimumTurn);
}

function EvaluateNormPower(normPower : float)
{
        if(normPower < 1)
                return 10 - normPower * 9;
        else
                return 1.9 - normPower * 0.9;
}

function GetGearState()
{
        var relativeVelocity : Vector3 = transform.InverseTransformDirection(rigidbody.velocity);
        var lowLimit : float = (currentGear == 0 ? 0 : gearSpeeds[currentGear-1]);
        return (relativeVelocity.z - lowLimit) / (gearSpeeds[currentGear - lowLimit]) * (1 - currentGear
* 0.1) + currentGear * 0.1;
}
```

Paso 8 Paso final - Habilitar Salto de entrada

Seleccionar el coche en la escena y en el componente de secuencia de comandos car.js en el inspector, seleccione la casilla de verificación para que sea Leap Enable.

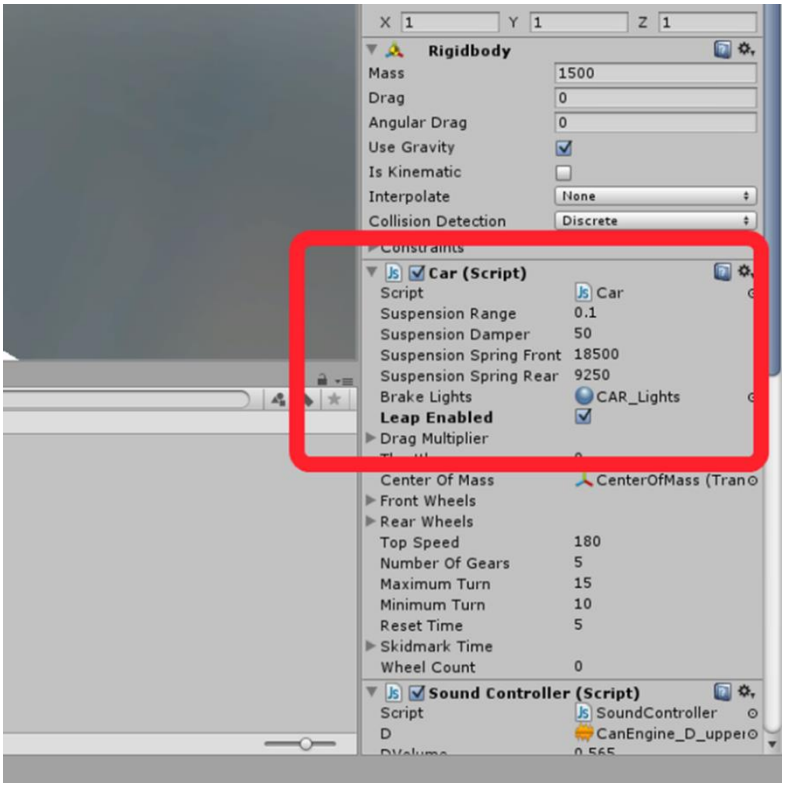


Figura 24 Habilitación Salto de movimiento